Remarks:

The Specification is amended to recite the reference numerals 20, 32, 33, 39, 152, 311, 312, 291, and 392, as appearing in Figures 1, 3A and 3B. The elements are clearly described in the Figures and Description in the Specification as originally filed. The Specification is amended for clarification only. Thus, there is no new subject matter.

Independent Claims 1, 9, 16 and 23 are amended. Claims 4 and 25 are canceled. Claims 1, 3, 5, 7-24 and 26-28 remain in the application.

ARGUMENT

Drawing Objections:

The Specification is amended herein to reference the reference numerals as identified by the Examiner. Therefore, this objection should be withdrawn.

Claim Objections:

The Examiner objects to Claim 23 and asserts that "executing on a host machine on the platform" should be shortened to read "executing on the platform." This suggestion shows a misunderstanding of the claim limitations. In a claimed embodiment of the invention, the platform comprises a virtual machine monitor (VMM), a virtual machine (VM) to execute the translated code, and a host environment running a host OS and a full-platform simulator. (See, at least, Figure 1 and paragraph [0016].) Claim 23 has been amended to clarify the terminology and be consistent with terminology used in the Specification. Claim 23 now requires *a host operating system executing in a host environment on the platform supports a full platform simulator that includes device models for the target processor, wherein the machine instructions are specific to the first instruction set architecture of the target processor, the virtual machine monitor executing translated code comprising sensitive instruction in an auxiliary simulator, and when the translated code are to access a simulated device, the full platform simulator executing the translated code that represents the device access.* It will be understood by one of skill in the art that in platforms having virtualization architecture, that a host machine means the same thing as a host environment running a host operating system. Therefore, this objection is moot.

<u>§ 112 Rejections:</u>

Claims 1, 3-5 and 7-28 are rejected under 35 U.S.C. § 112, first paragraph, as failing to comply with the written description requirement. The Examiner asserts that the claims contain subject matter which was not described in the specification to reasonably convey the invention to one of skill in the art. This rejection is respectfully traversed and Claims 1, 3-5 and 7-28 are believed allowable based on the foregoing and following discussion.

Claims 1 recited "the simulator executing the translated code that represents simulated operating system code to be executed on the virtual machine." Claims 9 and 16 recited "the simulator executing the translated code that represents simulated operating system code to be executed on a virtual machine." Claim 23 similarly recited "the simulator executing the translated code that represents simulated operating system code of the first instruction set architecture to be executed on the virtual machine in the second instruction set architecture." The Examiner asserts that these limitations do not appear in the specification as originally filed. Applicants respectfully disagree.

Claims 1, 9, 16 and 23 are amended to more clearly recite which translated instructions are to be executed by which simulation module. As shown in Figure 1, embodiments of the invention may use three distinct simulation modules: (1) simulated OS **152**; (2) auxiliary simulator **162**; and platform device simulator **12**. This embodiment is clearly shown in Figure 1, and supporting description as originally filed, illustrating a virtual machine 15 with translated code region 155 and a host environment 101 executing a host OS 11 and platform simulator 12. Beginning on page 3, paragraphs [0016] to [0018] it is described that:

> "The host environment 101 includes a host OS 11 and **<u>a full-platform simulator</u>** called SoftSDV (SOFTware-based Software Development Vehicle) 12. The SoftSDV 12 is a software simulator that **<u>simulates a PC hardware platform</u>**. The host environment 101 also includes a Direct Execution Driver 13, which serves as a gate between the host environment 101 and the direct execution environment 102.
>
> A user invokes the SoftSDV 12 in the host environment to **<u>execute a simulated OS 151 code</u>**. The SoftSDV 12 then passes simulation control to the direct execution environment 102 where the target CPU is simulated. **<u>When the target CPU accesses the simulated devices, the direct execution environment 102 passes simulation</u>**

> **control back to the SoftSDV 12.** After handling the simulated device access, the SoftSDV 12 again passes simulation control back to the direct execution environment 102.
>
> The direct execution environment 102 includes a virtual machine kernel (VMK) 14, a virtual machine (VM) 15, and a virtual machine monitor (VMM) 16. The VM 15 represents the target CPU. **The simulated OS 151 code runs in the VM 15. Most of the simulated instructions are executed directly (i.e., natively) on the host CPU. Those simulated instructions that access CPU system state, e.g., control registers, are intercepted and simulated in the VMM 16. Such instructions are called "sensitive instructions".** The VMM 16 monitors the VM 15 execution, and it provides the simulated OS 151 an illusion that the simulated OS controls all the platform resources." [emphasis added]

As is clearly described in the Specification, execution of the translated code depends on what type of instruction it is, on the target processor when an instruction require a device access, it is to be executed in the host environment using the full platform simulator. If the instruction is a "sensitive" instruction, as defined in the specification, it is to be executed by the VMM in the direct execution environment. Other instructions may be executed directly by the simulated OS in the VM, in the direct execution environment. The VMM controls the execution of the instructions based on the type of exception, if any, generated by an attempt of the VM to execute the translated code. Using the VMM and kernel to control execution of the translated code by capturing the exceptions allows the host machine to simulate execution of a first instruction set architecture even though the host machine utilizes a second instruction set architecture.

Only the instructions that do not compromise the integrity of the host OS are allowed to run natively. Instructions that access the privileged system state are intercepted and emulated in either the VMM simulator or the full platform simulator, for device access. A very efficient memory management scheme (based on segment virtualization) is put in place in order to allow instructions that access memory to run natively (with minimal management overhead).

§ 103 Rejections:

Claims 1, 3, 4, 9, 10, 11, 13, 16, 17, 18, 20 and 23-25 are rejected under 35 U.S.C. § 103(a) as being unpatentable over USPN 6,397,242 to Devine (hereinafter "Devine") in view of "*The Technology Behind Crusoe Processors*" by Alexander Klaiber (hereinafter "Klaiber) and further in view of view of "*Complete Computer System Simulation: The SimOS Approach*", Winter 1995, IEEE Parallel & Distributed Technology, pages 34-43, by Rosenblum et al. (hereinafter, "Rosenblum"). This rejection is respectfully traversed and Claims 1, 3, 4, 9, 10, 11, 13, 16, 17, 18, 20 and 23-25 are believed allowable as amended based on the foregoing and following discussion.

The Examiner now relies on Devine, as well as Klaiber, asserting that Devine teaches *the monitor modifying the <u>original values in a descriptor table</u>,* but that Klaiber teaches *to prevent the translated code from being accessed...* The Examiner asserts that because descriptor tables have a protection bit that modification to descriptor tables is taught by Devine. The Examiner asserts that Devine teaches that access to the translated code is prevented. However, modification of a protection bit in Devine, even combined with Klaiber's method of modifying actual page table will not result in the recited limitation. The virtualization of the segment information in the descriptor tables prevents the translated code from being accessed, but not because the page is "protected." Instead, the virtualization of the segments and the modification of the descriptor table enables the VMM to intercept each and every memory access.

Further, Claims 1, 9, 16 and 23 have been amended to clarify how the access is prevented. Claim 1, for instance, recites *a monitor executing in a direct execution environment on the host machine that translates the machine instructions into translated code, the monitor modifying original values of segment information in a descriptor table to force intervention by the monitor when the translated code references a memory access, thereby preventing the translated code from being accessed, and thereby preventing the translated code from being modified.* It should be noted that the translated code is prevented from being accesses and modified.

The specification, as originally filed describes that since "the code segment is fully virtualized, all attempts to perform memory access through the code segment should be intercepted. Thus all the instructions that have a code segment override prefix are detected by the VMM 16 during the code translation stage, and replaced with a capsule that causes an exit from the VM 15 to the VMM 16. The VMM 16 will simulate these instructions in the auxiliary ISA simulator 162 to avoid the translated code region from being accessed."

Descriptor tables, generally, are known in the art, as are segment tables/descriptors. However, use of the modification of the descriptor tables, as described in the specification, to prevent virtualized and translated code to be accessed is not taught or suggested by the cited prior art.

Moreover, Klaiber teaches write-protecting a page of memory. The modification as claimed by Applicants prevents the translated code from being accessed, not an entire page of memory. As described by Applicant, segmentation provides a mechanism for dividing the processor's linear address space into smaller protected regions called "segments." The operating system defines its segments by assigning a segment base, a segment limit, and different segment attributes, e.g., type, granularity, DPL (Descriptor Privilege Level). In contrast, pages are defined by the MMU. Thus, the prior art fails to show each limitation of the recited claims.

The Examiner further asserts that Rosenblum teach a full platform simulator, and combining the teachings of Rosenblum with Devine and Klaiber will result in Applicant's invention. Applicants respectfully disagree. Rosenblum seems to teach an operating system simulator. In contrast, the claimed simulator supports a full platform simulator which, as described in the specification, includes simulation of the instruction set architecture of the processor. Further, Rosenblum does not seem to teach a simulator that runs in a virtual machine.

The independent claims have been amended to more clearly recite the usage of descriptor tables and virtualization to trap execution of certain translated instructions, so that different simulation modules may be used for certain instructions. Even in combination, there is no motivation to integrate the simulator of Rosenblum into a platform running virtualized instructions in a virtual machine that is meant to simulate a target processor (other than the host processor). Moreover, it will not be easily apparent to one of skill in the art how one would apply the simulator of Rosenblum to the virtualization of Devine and the page locks of Klaiber. Moreover, it is not trivial to assume that the simulator of Rosenblum is scalable to operate in a virtualized environment.

Additionally, none of the references, either alone or in combination, teach that there are three simulation modules running on the platform to handle different categories of translated code. Thus, the Examiner has failed to present *prima facie* evidence of obviousness, and Claims 1, 9, 16 and 23 and their progeny are believed allowable.

The limitations of Claim 4 and 25 are added to Claims 1 and 23, respectively. The Examiner asserts that Devine appears to teach an auxiliary simulator. However, neither Devine, nor the other references teach that there are three simulator modules, and that the monitor intercepts memory accesses of the translated code such that different categories of instructions are executed by a specific one of three simulation modules, where one operates in the VM, one operates in the VMM and one operates in the host environment.

Regarding Claims 10 and 17, the Examiner fails to show that the simulator of Rosenblum would even operate properly in a virtualization environment. Thus, combining Rosenblum with Devine and Klaiber is not only non-obvious, but will not result in Applicants' claimed invention.

Claims 5, 12, 19 and 26 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Devine, Klaiber, Rosenblum and further in view of "*Running multiple operating systems concurrently on an IA32 PC using virtualization techniques*" by Kevin Lawton (hereinafter "Lawton"). This rejection is respectfully traversed and Claims 5, 12, 19 and 26 are believed allowable as amended based on the foregoing and following discussion.

Claims 5, 12, 19 and 26 are believed allowable, at least by being dependent on allowable base claims. Further, Lawton teach a method for trapping memory accesses so that the VMM will be forced to execute. However, Lawton does not teach that these traps may be deployed with translated code. Moreover, neither Devine, Klaiber nor Lawton teach that translated code is to be executed. The simulator of Rosenblum cannot be applied to the other references, as there is no suggestion that it might work in a virtualized system executing translated code. Thus, the Examiner has failed to provide a *prima facie* case of obviousness.

Claims 7, 8, 14, 15, 21-22 and 27-28 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Devine, Klaiber, Rosenblum and further in view of "*Operating systems internals and design principles*" by William Stallings (hereinafter "Stallings"). This rejection is respectfully traversed and Claims 7, 8, 14, 15, 21-22 and 27-28 are believed allowable as amended based on the foregoing and following discussion.

Claims 7, 8, 14, 15, 21-22 and 27-28 are believed allowable, at least by being dependent on allowable base claims.

Further, the Examiner asserts that Stallings teaches modifying the descriptor table to remove a portion of a segment that overlaps with the memory storing the translated code, or replace a segment to create a fault. However, it seems that Stallings merely teaches basic techniques for creating memory segments.

A cursory review of the teachings of Stallings do not show a teaching or suggestion that overlapping portions are to cause the descriptor table to be modified. It seems that Stallings is merely teaching how to set up a segmentation scheme and the translation of segment addresses. Stallings seems to teach allocating and translating segments and the handling of growing or shrinking data structures. There seems to be no teaching or suggestion for modifying the descriptor tables when segments overlap translated code. In fact, no mention of translated code seems to appear at all. Thus, Stallings fails to teach or suggest the claimed limitation.

The Examiner asserts that it would be obvious to modify Devine, Klaiber and Rosenblum with Stallings to modify a segment or replace a segment with a substitute segment to cause a fault. However, Stallings teaches only that more processes may be put in memory by overlaying them. This overlay scheme is based on original code, and not translated code. There is no suggestion in Stallings that translated code, i.e., code to be simulated, is to be written into data segments that cause an overlap of segments to the translated code. In contrast, the claimed invention translates machine instructions into translated code, and this translation may cause an overlap. The overlap here is not the same as overloading memory with too many processes in memory. The problem is different, the cause is different, and the result is different. Stallings will swap memory back and forth when a fault occurs.

In contrast, Applicants' invention transfers control to between a VM and VMM so that the translated code may be simulated. Applicants' claimed invention also simulates device access in the host environment. Thus, there is no suggestion in Stallings to replace segments for this purpose, nor would it be obvious to one of skill in the art. Moreover, as recited in the base claims, a modification of the segment information in the descriptor table is made to prevent the translated code from being accessed. This type of modification is neither taught nor suggested by Stallings. Moreover, as discussed above, Rosenblum cannot be combined with the virtualization and segmentation teachings, as there is no suggestion that Rosenblum's simulator would work in a virtualized system or that the simulator functions can be trifurcated, as described and claimed. Thus, all of the pending claims are believed allowable.

CONCLUSION

In view of the foregoing, Claims 1, 3, 5, 7-24 and 26-28 are all in condition for allowance. If the Examiner has any questions, the Examiner is invited to contact the undersigned at (703) 633-6845. Early issuance of Notice of Allowance is respectfully requested. Please charge any shortage of fees in connection with the filing of this paper, including extension of time fees, to Deposit Account 50-0221 and please credit any excess fees to such account.

Respectfully submitted,

Dated: 14 May 2008

/ Joni D. Stutman-Horn /
Joni D. Stutman-Horn, Reg. No. 42,173
Patent Attorney
Intel Corporation
(703) 633-6845

Intel Corporation
c/o Intellevate, LLC
P.O. Box 52050
Minneapolis, MN 55402